Neural Networks & Deep Learning

ICP-5

Sreeja Reddy Konda

700756597

GitHub-

https://github.com/SreejaReddyKonda/Neural-Network-Sreeja/blob/main/Neural%20Networks/ICP-5/ICP-5.ipynb

Video-

https://drive.google.com/file/d/1ICcRg8IXtlzDY-PHWicm3d6XKygqDeo6/view?usp=drive_link

1.

```
import numpy as np
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import SGD

# Fix random seed for reproducibility
seed = 7
np.random.seed(seed)

# Load data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
# Normalize inputs from 0-255 to 0.0-1.0
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
# One hot encode outputs
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
num_classes = y_test.shape[1]

# Original Model
original_model = Sequential()
original_model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same', activation='relu'))
original_model.add(Dropout(0.2))
original_model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
original_model.add(MaxPooling2D(pool_size=(2, 2)))
original_model.add(Flatten())
original_model.add(Dense(512, activation='relu'))
original_model.add(Dropout(0.5))
original_model.add(Dense(num_classes, activation='softmax'))
```

```
# Compile original model
original_epochs = 5
original_lrate = 0.01
original_decay = original_lrate / original_epochs
sgd = SGD(learning_rate=original_lrate, momentum=0.9, decay=original_decay, nesterov=False)
original_model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

# Fit original model
original_model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=original_epochs, batch_size=32)
# Evaluate original model
original_scores = original_model.evaluate(X_test, y_test, verbose=0)
original_accuracy = original_scores[1] * 100
print("Original Model Accuracy: %.2f%%" % original_accuracy)

# Modified Model
modified_model = Sequential()
modified_model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same', activation='relu'))
modified_model.add(Dropout(0.2))
modified_model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
modified_model.add(MaxPooling2D(pool_size=(2, 2)))
modified_model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
modified_model.add(Dropout(0.2))
modified_model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
modified_model.add(MaxPooling2D(pool_size=(2, 2)))
modified_model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
modified_model.add(Dropout(0.2))
modified_model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
modified_model.add(MaxPooling2D(pool_size=(2, 2)))
modified_model.add(Flatten())
modified_model.add(Dropout(0.2))
modified_model.add(Dense(1024, activation='relu'))
modified_model.add(Dropout(0.2))
modified_model.add(Dense(512, activation='relu'))
modified_model.add(Dropout(0.2))
modified_model.add(Dense(num_classes, activation='softmax'))
```

```
# Compile modified model
modified_epochs = 100
modified_lrate = 0.01
modified_decay = modified_lrate / modified_epochs
sgd = SGD(learning_rate=modified_lrate, momentum=0.9, decay=modified_decay, nesterov=False)
modified_model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

# Fit modified model
modified_model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=modified_epochs, batch_size=32)
# Evaluate modified model
modified_scores = modified_model.evaluate(X_test, y_test, verbose=0)
modified_accuracy = modified_scores[1] * 100
print("Modified Model Accuracy: %.2f%%" % modified_accuracy)

# Compare performances
performance_change = modified_accuracy - original_accuracy
print("Performance Change: %.2f%%" % performance_change)
```

Output-

```
Epoch 1/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.3034 - loss: 1.9197 - val_accuracy: 0.4387 - val_loss: 1.5407
Epoch 2/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.4827 - loss: 1.4368 - val_accuracy: 0.5604 - val_loss: 1.2392
Epoch 3/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.5570 - loss: 1.2435 - val_accuracy: 0.5812 - val_loss: 1.1737
Epoch 4/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.6137 - loss: 1.0839 - val_accuracy: 0.6172 - val_loss: 1.0886
Epoch 5/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.6650 - loss: 0.9488 - val_accuracy: 0.6387 - val_loss: 1.0358
Original Model Accuracy: 63.87%
Epoch 1/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 19s 9ms/step - accuracy: 0.2381 - loss: 2.0308 - val_accuracy: 0.4233 - val_loss: 1.5564
Epoch 2/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4649 - loss: 1.4632 - val_accuracy: 0.5276 - val_loss: 1.3230
Epoch 3/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.5568 - loss: 1.2366 - val_accuracy: 0.6119 - val_loss: 1.0880
Epoch 4/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.6177 - loss: 1.0722 - val_accuracy: 0.6522 - val_loss: 0.9982
Epoch 5/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.6603 - loss: 0.9502 - val_accuracy: 0.6715 - val_loss: 0.9288
Epoch 6/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.6976 - loss: 0.8656 - val_accuracy: 0.6986 - val_loss: 0.8512
Epoch 7/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 9s 6ms/step - accuracy: 0.7207 - loss: 0.7958 - val_accuracy: 0.7269 - val_loss: 0.7923
Epoch 8/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.7413 - loss: 0.7418 - val_accuracy: 0.7432 - val_loss: 0.7463
Epoch 9/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.7500 - loss: 0.6988 - val_accuracy: 0.7442 - val_loss: 0.7344
Epoch 10/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.7654 - loss: 0.6680 - val_accuracy: 0.7484 - val_loss: 0.7326
Epoch 11/100
```

```
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.4718 - loss: 1.5702 - val_accuracy: 0.5123 - val_loss: 1.4548
Epoch 86/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4813 - loss: 1.5476 - val_accuracy: 0.4517 - val_loss: 1.5552
Epoch 87/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.4648 - loss: 1.5703 - val_accuracy: 0.4439 - val_loss: 1.6448
Epoch 88/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - accuracy: 0.4815 - loss: 1.5368 - val_accuracy: 0.4622 - val_loss: 1.5871
Epoch 89/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.4740 - loss: 1.5524 - val_accuracy: 0.3331 - val_loss: 1.8838
Epoch 90/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4607 - loss: 1.5915 - val_accuracy: 0.4267 - val_loss: 1.6445
Epoch 91/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.4860 - loss: 1.5198 - val_accuracy: 0.4459 - val_loss: 1.6053
Epoch 92/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 9s 6ms/step - accuracy: 0.4560 - loss: 1.5939 - val_accuracy: 0.4173 - val_loss: 1.6650
Epoch 93/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.4505 - loss: 1.6068 - val_accuracy: 0.3696 - val_loss: 1.7784
Epoch 94/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.4518 - loss: 1.6136 - val_accuracy: 0.4424 - val_loss: 1.6301
Epoch 95/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - accuracy: 0.4535 - loss: 1.6133 - val_accuracy: 0.4486 - val_loss: 1.5932
Epoch 96/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.4748 - loss: 1.5497 - val_accuracy: 0.2292 - val_loss: 2.0976
Epoch 97/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 9s 6ms/step - accuracy: 0.4159 - loss: 1.6989 - val_accuracy: 0.4667 - val_loss: 1.5389
Epoch 98/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 9s 6ms/step - accuracy: 0.4489 - loss: 1.6157 - val_accuracy: 0.4958 - val_loss: 1.4874
Epoch 99/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - accuracy: 0.4306 - loss: 1.6493 - val_accuracy: 0.4672 - val_loss: 1.5532
Epoch 100/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.4572 - loss: 1.5880 - val_accuracy: 0.4243 - val_loss: 1.6429
Modified Model Accuracy: 42.43%
Performance Change: -21.44%
```

2.

```
import matplotlib.pyplot as plt

# ... [previous model training code] ...

# Predict the first 4 images of the test data
predictions = modified_model.predict(X_test[:4])
predicted_classes = np.argmax(predictions, axis=1)
actual_classes = np.argmax(y_test[:4], axis=1)

# Display predictions and actual classes
print("Predicted Classes:", predicted_classes)
print("Actual Classes:", actual_classes)

# Visualize the first 4 images with their predictions
plt.figure(figsize=(10, 5))
for i in range(4):
    plt.subplot(2, 4, i + 1)
    plt.imshow(X_test[i])
    plt.title(f'Pred: {predicted_classes[i]}, Actual: {actual_classes[i]}')
    plt.axis('off')
plt.show()
```

Output-

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 275ms/step
Predicted Classes: [5 8 8 0]
Actual Classes: [3 8 8 0]
```

3.

```
# Fit modified model and store the history
history = modified_model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=modified_epochs, batch_size=32)

# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

plt.tight_layout()
plt.show()
```

Output-

```
[10] Epoch 97/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4412 - loss: 1.6083 - val_accuracy: 0.4724 - val_loss: 1.5143
Epoch 98/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.4276 - loss: 1.6524 - val_accuracy: 0.4964 - val_loss: 1.4663
Epoch 99/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4675 - loss: 1.5405 - val_accuracy: 0.4613 - val_loss: 1.5311
Epoch 100/100
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.4530 - loss: 1.5806 - val_accuracy: 0.4320 - val_loss: 1.6196
```